```
   outp(addr, 0x8d);
 }
/***************************** ********** ******** *******************************/
/*                                                   */
/* NAME          : GET_WEIGHT - Get the current weight                   */
/* AUTHOR        : Celestine Vettical                       */
/* DATE WRITTEN  : 05-Nov-1990                         */
/* DATE REVISION :                             */
/* PURPOSE       : To provide a procedure to get the current weight on a   */
/*               given scale in counts.                   */
/* MODEL         : This procedure uses direct control register accessing  */
/*               using the library calls inp and outp to get the count  */
/* VERSION       : 1.1  (Release 1,   Version 1)                */
/* HISTORY       : NUMBER   DATE          DESCRIPTION               */
/*               Original  05-Nov-90   Designer Original Release        */
/* AGREEMENTS    : Development by: Designer (05-Nov-90)              */
/*               Used by: Designer in the sequential ZIPLUS program     */
/* REQUIREMENTS  : To provide a C interface for the scale board.           */
/* DEPENDENCIES   : Includes serial.h -- a definition file for sequential  */
/*              procedures for ZIPSTER PLUS                     */
/* PARAMETERS    : NAME        DESCRIPTION              UNITS        */
/*              scale      The scale select control   integer      */
/*                         register address(SCALEA or            */
/*                         SCALEB defined in serial.h)          */
/* ABSTRACT      : This procedure can be used to get the current weight   */
/*              in counts.                          */
/* PERFORMANCE   : Unknown                                */
/* RESTRICTIONS  : The A to D scale board should be set to the base       */
/*              addresses given in "serial.h"             */
/* ERRORS PROPAGATED: status = Valid if zero, else scale is unstable       */
/* ERRORS HANDLED : None                                  */
/* SAMPLE CALL   : get_weight(SCALEA)                        */
/*                                                  */
/*****************************************************************************/
/*  Copyright (c) 1990                               */
/*  Pi Electronics Corp.                             */
/*  9777 W Gulf Bank Rd                                */
/*  Houston, Texas  77040-3113                            */
/*  (713) 896-5800                                   */
/*  ALL RIGHTS RESERVED                                  */
/*****************************************************************************/


/*****************************************************************************/
/* read_scalereg(reg_num):   Read Scale Board Data                 */
/*                                                  */
/* Function to read a register from the scale. Passed argument is the      */
/* register number to be input.                          */
/*                                                  */
/* Return:      value input from scale board, char.                 */
/*****************************************************************************/
unsigned char read_scalereg(reg_num)
unsigned char reg_num
{
  while (inp(REG_STATUS) & DEV_BUSY);        /* be sure it isnt busy   */
  outp(REG_COMMAND, reg_num);              /* select the register    */
  while (inp(REG_STATUS) & DEV_BUSY);        /* wait for not busy      */
  return(inp(REG_CONTROL));               /* return control reg value*/
}
/*****************************************************************************/
/* write_scalereg(reg_num,regdata);   Write Scale Board Data           */
/*                                                  */
/* Function to write a register from the scale. Passed argument is the      */
/* register number to be written and the data to write to it.          */
/*                                                  */
/* Return:      nothing.                                */
/*****************************************************************************/
void write_scalereg(reg_num,regdata)
unsigned char reg_num,regdata;
{
  while (inp(REG_STATUS) & DEV_BUSY);        /* be sure it isnt busy   */
  outp(REG_COMMAND, reg_num);              /* select the register    */
  while (inp(REG_STATUS) & DEV_BUSY);        /* wait for not busy      */
  outp(REG_CONTROL,regdata);              /* update the control reg  */
  return;
```

```
}
/*****************************************************************************/
void __export FAR PASCAL init_scale(void)
{

  /* initialize the scale board operating parameters      */
  write_scalereg(SEL_CHA_SCAN_RATE,CHA_SCAN_RATE);  /* update scan rate     */
  write_scalereg(SEL_CHA_DEAD_BAND,CHA_DEADBAND);   /* update the deadband  */
  write_scalereg(SEL_CHA_SMOOTH,CHA_SMOOTH_COEF);   /* update smooth coeff. */
  write_scalereg(SEL_CHB_SCAN_RATE,CHB_SCAN_RATE);  /* update scan rate     */
  write_scalereg(SEL_CHB_DEAD_BAND,CHB_DEADBAND);   /* update the deadband  */
  write_scalereg(SEL_CHB_SMOOTH,CHB_SMOOTH_COEF);   /* update smooth coeff. */

      /* Read weight calibration constants */
  A_Cal_factor = (unsigned int)(read_scalereg(SEL_CHA_MSB_CAL) << 8) +
           read_scalereg(SEL_CHA_LSB_CAL);
  B_Cal_factor = (unsigned int)(read_scalereg(SEL_CHB_MSB_CAL) << 8) +
           read_scalereg(SEL_CHB_LSB_CAL);
  A_Null_weight = (unsigned int)(read_scalereg(SEL_CHA_MSB_NUL) << 8) +
            read_scalereg(SEL_CHA_LSB_NUL);
  B_Null_weight = (unsigned int)(read_scalereg(SEL_CHB_MSB_NUL) << 8) +
            read_scalereg(SEL_CHB_LSB_NUL);

  /**** avoid zero divide when scale is not calibrated!!!    */
  if (A_Cal_factor == 0) A_Cal_factor = 1;
  if (B_Cal_factor == 0) B_Cal_factor = 1;
}
/*****************************************************************************/
/* Function to read a stable weight in counts from the given scale         */
/* Return value: 0 -> successful                                  */
/*              1 -> unsuccessful (not stable)                    */
/*****************************************************************************/
unsigned char get_weight(weight, scale_num)
unsigned int FAR *weight;
unsigned char scale_num;
{
  unsigned long start_time;
  unsigned char stable, scale;

  while (inp(REG_STATUS) & DEV_BUSY);    /* be sure the scale isnt busy   */

  if (scale_num == 1)  /* letter scale */
  {
    scale  = SCALEA;
    stable = CHA_STABLE;
    outp(REG_COMMAND,SCALEA);
  }
  else
  {
    scale  = SCALEB;
    stable = CHB_STABLE;
    outp(REG_COMMAND,SCALEB);
  }

  start_time = GetTickCount();
  while (inp(REG_STATUS) & DEV_BUSY);
```

```
    while ( !(inp(REG_STATUS) & stable) && (start_time > GetTickCount() - 1000) )
      ;   /* Read status and wait until stable reading and not busy */

    *weight = inpw(REG_DATA);

    if (inp(REG_STATUS) & stable)
    {           /* delay 1/4 second to see that stable remains  */
      start_time = GetTickCount();
      while ( (inp(REG_STATUS) & stable) && (start_time > GetTickCount() - 250) );
      }

    if (inp(REG_STATUS) & stable)
      return(0);
    else
      return(1);
}
/**************************************************************************/
/* ZERO_SCALE  : Function to zero the scales                             */
/*   Return Value :   0 -> successfull                          */
/*                    1 -> no stable reading                  */
/*                    2 -> letter scale not empty              */
/**************************************************************************/
unsigned char __export FAR PASCAL zero_scale(scale, changeZero)
unsigned char scale;
unsigned char changeZero;
{
    int loop_count=0, broke_loop_count=1;
    unsigned int cur_tare, null_wgt;

    if (scale == 1 )  /* letter scale */
      null_wgt = A_Null_weight;
    else
      null_wgt = B_Null_weight;

    for(;;)
    {
      loop_count = 0;
      while (get_weight(&cur_tare, scale) != 0)
      {
        if (loop_count ++ == 200)  /* no stable reading after 200 reads */
          return(1);
      }
      if (get_weight(&cur_tare, scale) == 0)  /* 2 succesive stable reading */
      {
        if (abs(cur_tare - null_wgt) < 40)
           break;
        if (scale == 1)
        {
          if (changeZero == 1)
            null_wgt = cur_tare;
          else
            return(2);
        }
        else
          null_wgt = cur_tare;
      }
    }

    if (scale == 1)
    {
      A_Null_weight = cur_tare;
      write_scalereg(SEL_CHA_MSB_NUL,(unsigned char)(A_Null_weight >> 8) );
      write_scalereg(SEL_CHA_LSB_NUL,(unsigned char)A_Null_weight);
    }
    else
    {
/*      get_fine_weight(&cur_tare, scale, 5);*/
      B_Null_weight = cur_tare;
      write_scalereg(SEL_CHB_MSB_NUL,(unsigned char)(B_Null_weight >> 8) );
      write_scalereg(SEL_CHB_LSB_NUL,(unsigned char)B_Null_weight);
    }
    return(0);
}
/**************************************************************************/
/* Function to read a stable weight in counts from the given scale        */
/* Return value:  0 -> stable weight counts                    */
/*             -1 -> unsuccessful (not stable)               */
/*            +ve -> stable real weight ( when display = 1)        */
/**************************************************************************/
double __export FAR PASCAL find_weight(scal_num, calculated_weight, display)
unsigned char scal_num;
char FAR *calculated_weight;
unsigned char display;
{
  unsigned int wt_cnt;
  double wt_lb, wt_oz;
  char wt_str[10], oz_str[10];
  double oz_part, lb_part;
  unsigned int cal_factor, null_wgt;


  if (get_weight(&wt_cnt, scal_num) == 0)  /* stable reading */
  {
    if (display == 0)  /* no need to find display weight */
      return(0);
    else  /* calculate real weight */
    {
      if (scal_num == 1)  /* letter scale */
      {
        cal_factor = A_Cal_factor;
        null_wgt = A_Null_weight;
      }
      else
      {
        cal_factor = B_Cal_factor;
        null_wgt = B_Null_weight;
      }
      wt_lb = wt_cnt - null_wgt;
      if (wt_lb > 60000)
        wt_lb = 0;              /* below null reading, set to zero  */
      else
```

```
     wt_lb = wt_lb/cal_factor;
    wt_oz = wt_lb*16;

      /* Rate Classifier Mode Display */
    if (wt_oz <= 16.0)   /* less than 1 lb. incl. */
      wt_oz = wt_oz - 0.03;   /* subtract the maintenance tolerance */
    else if (wt_oz <= 64.0)    /* less than 4 lb. incl. */
      wt_oz = wt_oz - 0.12;   /* subtract the maintenance tolerance */
    else if (wt_oz <= 112.0)   /* less than 7 lb. incl. */
      wt_oz = wt_oz - 0.2;    /* subtract the maintenance tolerance */
    else                      /* less than 25lb.              */
      wt_oz = wt_oz - 0.4;    /* subtract the maintenance tolerance */

    if (wt_oz < 0)   /* avoid negative display */
      wt_oz = 0.0;

/*-----------------------------------------------*/
/*  I am using manual_fcvt instead of the        */
/*  wsprintf function for floating point numbers. */
/*-----------------------------------------------*/

 //    wsprintf(wt_str, "%6.2f", wt_oz);

    manual_fcvt(wt_oz, 6, 2, (LPSTR) wt_str);

    if (wt_oz <= 32.0)   /* less than or equal to 2 lb.  */
    {
      if( (wt_str[5]-'0') <5) && ((wt_str[5]-'0') !=0) )
      {
        wt_str[5] = '5';
        wt_oz = manual_atof(wt_str);
      }
      else if( (wt_str[5]-'0') >5 )
      {
        wt_str[5] = '0';
        wt_str[4] = wt_str[4] +1;
        if( (wt_str[4]-'0') > 9)
        {
          wt_str[4] = '0';
          wt_oz = manual_atof(wt_str) + 1.0;
        }
        else
          wt_oz = manual_atof(wt_str);
      }

    else if (wt_oz <= 112.0)       /* less than 7 lb.   */

      if( (wt_str[5]-'0') >0 )
      {
        wt_str[5] = '0';
        wt_str[4] = wt_str[4] +1;
        if( (wt_str[4]-'0') > 9)
        {
          wt_str[4] = '0';
          wt_oz = manual_atof(wt_str) + 1.0;
        }
```

```
            else
              wt_oz = manual_atof(wt_str);
          }
          else
            wt_oz = manual_atof(wt_str);
      }
      else  /* over 7 lb. */
      {
        if( (wt_str[5]-'0') >0 )
        {
          wt_str[5] = '0';
          wt_str[4] = wt_str[4] +1;
          if( (wt_str[4]-'0') > 9)
          {
            wt_str[4] = '0';
            wt_oz = manual_atof(wt_str) + 1.0;
//            wsprintf(wt_str, "%6.2f", wt_oz);
            manual_fcvt(wt_oz, 6, 2, (LPSTR) wt_str);
          }
        }
        if( (wt_str[4]-'0') >0 )
        {
          wt_str[4] = wt_str[4] + ((wt_str[4]-'0')%2);
          if( (wt_str[4]-'0') > 9 )
          {
            wt_str[4] = '0';
            wt_oz = manual_atof(wt_str) + 1.0;
          }
          else
            wt_oz = manual_atof(wt_str);
        }
        else
          wt_oz = manual_atof(wt_str);
    }

    if ( wt_oz <= 0.05)
      wt_oz = 0.0;

    wt_lb = wt_oz/16.0;

//  oz_part = modf(wt_lb, &lb_part);

    /*------ -------- --------------------------------*/
    /* A manual way of performing the modf function.  */
    /*------- ------------------------------------------*/
    lb_part = (double) ((int)wt_lb);
    oz_part = wt_lb-lb_part;    // NOTE:  Don't need this statement
                                //        because of next statement

    oz_part = (wt_lb - lb_part)*16 ;

//  wsprintf(calculated_weight, "%2d lb %5.2f oz", (int)lb_part, oz_part);
    manual_fcvt(oz_part, 5, 2, (LPSTR) oz_str);
    wsprintf(calculated_weight, "%2d lb %s oz", (int)lb_part, (LPSTR) oz_str);
    if (scal_num == 1) /* letter scale */
      return(wt_oz);
      else
        return(wt_lb);
    }
  }
  else
    return(-1);
}
/*****************************************************************************/
/*---------------------------------------------------*/
/*                                                   */
/*  manual_fcvt is a float conversion procedure.     */
/*  The parameters are:                              */
/*                                                   */
/*    Float_Value  - the value to convert to a string. */
/*                                                   */
/*    Digits       - the total number of characters in */
/*                   the string, including the decimal */
/*                   point and sign.                 */
/*                                                   */
/*    Precision    - the number of digits after the  */
/*                   decimal point to represent.     */
/*                                                   */
/*    Float_String - the result string.  It must be  */
/*                   memory set asside by the calling */
/*                   program.                        */
/*                                                   */
/*  There is one known limitation: the number of digits */
/*  of resolution (including the digits before the   */
/*  decimal point) must not exceed 38.  The subscript */
/*  of the local char array digits_str can be modified */
/*  as needed for this situation.                    */
/*                                                   */
/*---------------------------------------------------*/

void manual_fcvt(double Float_Value, int Digits, int Precision,
              LPSTR Float_String)
{
 char digits_str[40];
 long digits, precision_multiplier=1L, int_part, float_part;
 int i;

  for (i=0;i<Precision;i++)
   precision_multiplier *= 10L;

 if (Float_Value*(double)precision_multiplier<0.0)
   digits = (long) (Float_Value*(double)precision_multiplier - 0.5);
 else
   digits = (long) (Float_Value*(double)precision_multiplier + 0.5);

 if (digits<0L) {
   lstrcpy(Float_String, "-");
   digits = -digits;
 }
 else
   lstrcpy(Float_String, "");
  int_part = digits/precision_multiplier;
  float_part = digits-int_part*precision_multiplier;

  wsprintf(digits_str, "%d", int_part);
  lstrcat(Float_String, digits_str);

  lstrcat(Float_String, ".");

  wsprintf(digits_str, "%d", float_part);
  lstrcat(Float_String, digits_str);

  if (lstrlen(Float_String)<Digits) {
    lstrcpy(digits_str, " ");
    for (i=1;i<Digits-lstrlen(Float_String);i++)
      lstrcat(digits_str, " ");
    lstrcat(digits_str, Float_String);
    lstrcpy(Float_String, digits_str);
  }
}
/********************************************************************************/
/*-----------------------------------------------------*/
/*                                                     */
/*  manual_atof work the same as the C function atof.  */
/*                                                     */
/*-----------------------------------------------------*/

double manual_atof(LPSTR Float_String)
{
 int i=0, len, done=0;
 double ret_val=0.0, dec_val=1.0, neg=1.0;

 len = lstrlen(Float_String);

 while (Float_String[i]==' ' && i<len)
   i++;

 if (i>=len)
   return ret_val;

 if (Float_String[i]=='-') {
   neg= 1.0;
   i++;
 }

 while (Float_String[i]!='.' && i<len) {

   if (Float_String[i] < '0' || Float_String[i]>'9')
     return neg*ret_val;

   ret_val = 10.0*ret_val+(double)(Float_String[i]-'0');
   i++;
 }

 if (i>=len)
   return neg*ret_val;
  i++;  // Skip the decimal point

  while (i<len) {

    if (Float_String[i] < '0' || Float_String[i]>'9')
      return neg*ret_val;

    dec_val = dec_val/10.0;
    ret_val = ret_val+((double)(Float_String[i]-'0'))*dec_val;
    i++;
  }

  return neg*ret_val;
}
/**********************************************************************************/
```